Amendments to the Claims

1. (Currently Amended) A method of instrumenting a program to provide instrumentation data for detecting memory leaks in the program, the method comprising:

creating an instrumented version of the program comprising duplicate versions of at least some code paths in the program[[s]], such that a duplicate code path has an original version code path and an instrumented version code path with instrumentation code for capturing instrumentation data;

tracking a frequency of execution of the code paths;

when a code path is to be executed, determining to dispatch execution into the instrumented version code path at a sampling rate for the respective code path and otherwise into the original version code path such that, for a given sampling rate, a ratio of a number of executions of the instrumented version code path to a total number of executions of both the instrumented version code path and the original version code path is equivalent to the given sampling rate; ~~and~~

adapting the sampling rate for the code paths according to the frequency of execution of the code paths, such that, after adapting, [[a]] the ratio of a number of executions of the instrumented version code path to a total number of executions of the code path is equivalent to the ~~adjusted~~ adapted sampling rate, wherein the sampling rate for executing the code paths is adapted such that the adapted sampling rate is inversely related to the frequency of execution of the code paths;

storing instrumentation data obtained by execution of the instrumented version of the software; and

reporting all objects that, according to the instrumentation data, satisfy a predefined staleness condition as memory leaks, wherein the predefined staleness condition comprises determining whether an object in the heap has been accessed within a predetermined length of time.

2. (Original) The method of claim 1 wherein instrumentation data comprises data relating to runtime data references, branch executions, memory allocations, synchronization events, data loads, data stores, or branches.

3.-6. (Cancelled)

7. (Currently Amended) A method for bursty tracing to detecting memory leaks in software by adapting a sampling rate of executing instrumented procedures in the software during runtime, the method comprising:

creating an instrumented version of the software containing an original version and an instrumented copy version of each procedure in the software, wherein only one version of the each procedure will be executed each time when the each procedure is required to execute during the runtime;

executing the instrumented version of the software;

tracking frequency of execution of the procedures, wherein the instrumented copy version of the procedures are sampled at rates adjusted to be inversely related to the frequency of execution of the procedures, including:

sampling at higher rates for the procedures whose total number of executions of both the original versions and the copy versions are executed less frequently, and sampling ~~sampled~~ at lower rates for procedures whose total number of execution of both the original versions and the copy versions are executed more frequently, wherein [[a]] the sampling rate for a given procedure comprises a number of executions of the instrumented version of the procedure taken as percentage of total number of executions of both versions of the procedure;

storing instrumentation data obtained by execution of the instrumented version of the software; and

reporting all objects that satisfy a predefined staleness condition as memory leaks, wherein the predefined staleness condition comprises determining whether an object in the heap has been accessed within a predetermined length of time.

8. (Original) The method of claim 7 wherein instrumentation data comprises heap allocation, heap free and heap access information.

9. (Original) The method of claim 7 wherein reporting all objects comprises reporting the heap object, responsible allocation, heap frees that deallocated objects created at that allocation site, and the last access to the leaked object.

10. (Original) The method of claim 9 wherein a source code browser highlights the last access to a leaked object.

11.-25. (Cancelled)

26. (New) The method of claim 1 wherein instrumentation data comprises heap allocation, heap free, and heap access information.

27. (New) The method of claim 1 wherein reporting all objects comprises reporting the heap object, responsible allocation, heap frees that deallocated objects created at that allocation site, and the last access to the leaked object.

28. (New) The method of claim 27 wherein a source code browser highlights the last access to a leaked object.

29. (New) A method of instrumenting a program to provide instrumentation data for detecting data races in the program, the method comprising:

creating an instrumented version of the program comprising duplicate versions of at least some code paths in the program, such that a duplicate code path has an original version code path and an instrumented version code path with instrumentation code for capturing instrumentation data;

tracking a frequency of execution of the code paths;

when a code path is to be executed, determining to dispatch execution into the instrumented version code path at a sampling rate for the respective code path and otherwise into the original version code path such that, for a given sampling rate, a ratio of a number of executions of the instrumented version code path to a total number of executions of both the

instrumented version code path and the original version code path is equivalent to the given sampling rate; and

adapting the sampling rate for the code paths according to the frequency of execution of the code paths, such that, after adapting, the ratio of a number of executions of the instrumented version code path to a total number of executions of the code path is equivalent to the adapted sampling rate, wherein the sampling rate for executing the code paths are adapted at a rate inversely related to the frequency of execution of the code paths;

storing instrumentation data obtained by execution of the instrumented version of the software; and

reporting variables, that, according to the instrumentation data, satisfy a predefined data race condition based on lock acquisitions and releases for the variables.

30. (New) The method of claim 29, wherein instrumentation data comprises sets of locks for variables in the software and sets of locks held by threads which access shared variables.

31. (New) The method of claim 30, wherein storing instrumentation data comprises, when a thread accesses a shared variable, refining a set of locks for the variable by intersecting it with a set of locks for the thread.

32. (New) The method of claim 31, wherein the predefined data race condition comprises determining, when an intersection is made between a set of locks for a variable and a set of locks for a thread, whether the intersection is null.

33. (New) A method for bursty tracing to detect data races in software by adapting a sampling rate of executing instrumented procedures in the software during runtime, the method comprising:

creating an instrumented version of the software containing an original version and an instrumented copy version of each procedure in the software, wherein only one version of the each procedure will be executed each time when the each procedure is required to execute during the runtime;

executing the instrumented version of the software;

tracking frequency of execution of the procedures, wherein the instrumented copy version of the procedures are sampled at rates adjusted to be inversely related to the frequency of execution of the procedures, including: sampling at higher rates for the procedures whose total number of executions of both the original versions and the copy versions are executed less frequently, and sampling at lower rates for procedures whose total number of execution of both the original versions and the copy versions are executed more frequently, wherein the sampling rate for a given procedure comprises a number of executions of the instrumented version of the procedure taken as percentage of total number of executions of both versions of the procedure;

storing instrumentation data obtained by execution of the instrumented version of the software; and

reporting variables that, according to the instrumentation data, satisfy a predefined data race condition based on lock acquisitions and releases for the variables.

34. (New) The method of claim 29, wherein instrumentation data comprises sets of locks for variables in the software and sets of locks held by threads which access shared variables.

35. (New) The method of claim 30, wherein storing instrumentation data comprises, when a thread accesses a shared variable, refining a set of locks for the variable by intersecting it with a set of locks for the thread.

36. (New) The method of claim 31, wherein the predefined data race condition comprises determining, when an intersection is made between a set of locks for a variable and a set of locks for a thread, whether the intersection is null.

37. (New) A method of instrumenting a program to provide instrumentation data for checking correctness of memory accesses in the program, the method comprising:

creating an instrumented version of the program comprising duplicate versions of at least some code paths in the program, such that a duplicate code path has an original version code path

and an instrumented version code path with instrumentation code for capturing instrumentation data;

tracking a frequency of execution of the code paths;

when a code path is to be executed, determining to dispatch execution into the instrumented version code path at a sampling rate for the respective code path and otherwise into the original version code path such that, for a given sampling rate, a ratio of a number of executions of the instrumented version code path to a total number of executions of both the instrumented version code path and the original version code path is equivalent to the given sampling rate; and

adapting the sampling rate for the code paths according to the frequency of execution of the code paths, such that, after adapting, the ratio of a number of executions of the instrumented version code path to a total number of executions of the code path is equivalent to the adapted sampling rate, wherein the sampling rate for executing the code paths are adapted at a rate inversely related to the frequency of execution of the code paths;

storing instrumentation data obtained by execution of the instrumented version of the software; and

reporting memory accesses, that, according to the instrumentation data, access memory locations incorrectly.

38. (New) The method of claim 37, wherein reporting memory accesses comprises reporting checking invariance in the software.

39. (New) The method of claim 38, wherein the instrumentation data comprises array accesses.

40. (New) The method of claim 39, wherein checking invariance comprises checking the array accesses against array bounds.

41. (New) A method for bursty tracing to detect incorrect memory accesses in software by adapting a sampling rate of executing instrumented procedures in the software during runtime, the method comprising:

creating an instrumented version of the software containing an original version and an instrumented copy version of each procedure in the software, wherein only one version of the each procedure will be executed each time when the each procedure is required to execute during the runtime;

executing the instrumented version of the software;

tracking frequency of execution of the procedures, wherein the instrumented copy version of the procedures are sampled at rates adjusted to be inversely related to the frequency of execution of the procedures, including: sampling at higher rates for the procedures whose total number of executions of both the original versions and the copy versions are executed less frequently, and sampling at lower rates for procedures whose total number of execution of both the original versions and the copy versions are executed more frequently, wherein the sampling rate for a given procedure comprises a number of executions of the instrumented version of the procedure taken as percentage of total number of executions of both versions of the procedure;

storing instrumentation data obtained by execution of the instrumented version of the software; and

reporting memory accesses, that, according to the instrumentation data, access memory locations incorrectly.

42. (New) The method of claim 41, wherein reporting memory accesses comprises reporting checking invariance in the software.

43. (New) The method of claim 42, wherein the instrumentation data comprises array accesses.

44. (New) The method of claim 43, wherein checking invariance comprises checking the array accesses against array bounds.